

# MANIFOLD

*Last updated:* Tue Feb 18 15:00:00 MET DST 1997 by [Juan Carlos Cruz](#)

[ [Top](#) | [Summary](#) | [Component Model](#) | [Usage](#) | [Discussion](#) | [References](#) ]

---

## Summary

**Manifold** is a coordination language for writing program modules (coordinator processes) to manage complex, dynamically changing interconnections among sets of independent, concurrent, cooperating processes that comprise a single application.

[ [Top](#) | [Summary](#) | [Component Model](#) | [Usage](#) | [Discussion](#) | [References](#) ]

---

## Component Model

[ [Definition](#) | [Interface](#) | [Encapsulation](#) | [Granularity](#) | [Generality](#) | [Binding](#) | [Semantics](#) ]

### Component Definition

The conceptual model behind **Manifold** is based on the IWIM model described in [2]. The basic concepts in the IWIM model are *processes*, *events*, *ports*, and *channels*. A process is a *black box* with well defined ports of connection through which it exchanges *units* of information with other processes in its environment. The scope of a port identifier is restricted to the process that declares it. Units of information are exchanged using standard I/O type primitives analogous to read and write. Each port is used for the exchange of information in only one direction: either into (input port) or out of (output port) a process.

Notation:  $p.i$  refers to the port  $i$  of a process  $p$ .

The interconnections between the ports of processes are made through channels. A channel connects a (port of a) producer (process) to a (port of a) consumer (process).

Notation:  $p.o \rightarrow q.i$  denotes a channel connecting the port  $o$  of the producer process  $p$  to the port  $i$  of the consumer process  $q$ .

Independent of the channels, there is an event mechanism for

information exchange in IWIM. Events are broadcast by their sources in their environment, yielding and event occurrence. In principle any process in an environment can pick up a broadcast event occurrence. In practice, usually a few number of processes pick up occurrences of each event, because only they are *tuned in* to their sources.

The IWIM model supports *anonymous communication*: in general, a process does not, and need not, know the identity of the processes with which it exchanges information, reducing in this way the dependency of a process on its environment and making processes more reusable.

A Process in IWIM can be regarded as a worker process or a manager (coordinator) process. The responsibility of a worker process is to perform a (computational) task. A worker process is not responsible for the communication that is necessary for it to obtain the proper input it requires to perform its task, nor is it responsible for the communication that is necessary to deliver the results it produces to their proper recipients. It is always the responsibility of a manager process to arrange for and to coordinate the necessary communications among a set of worker processes.

Primitives for a Worker: *read, write, raise (i.e. raise an event), and pickup (i.e. pick up an event).*

Primitives for a manager: *create new instances of processes, broadcast and react to event occurrences, destroy channel (re) connections between various ports of the processes instances it knows.*

## **Interface**

The workers of the IWIM model are called atomic processes in Manifold. Any operating system-level process can be used as an atomic process in Manifold. Atomic process can only produce and consume units through their ports, raise and receive events and compute. Manager/Coordinator processes are written in the Manifold language and are called manifolds. A manifold definition (i.e. the definition of a coordinator process in this language) consists of a header and a body. The header of a manifold (or atomic process definition) gives its name, the number and types of its parameters, and the names of its input and output ports. Parameters can be manifold definitions, processes, events, and ports. The body of a manifold definition is a block. A block consists of a finite number of states. Each state has a label and a body. The label of a state defines the condition under which a transition to that state is possible, in terms of a conjunction of patterns related with event occurrences. The body of a simple state defines the set of actions that are to be performed upon transition to that state.

## **Encapsulation Boundary**

There are two type of components workers and managers (coordinators). Workers can produce and consume units of information through their ports, raise and receive events and compute. Managers can create new instances of processes and broadcast and react on event occurrences. They can also create and destroy channel (re) connections between various ports of the process instances it knows.

## **Granularity**

Components are processes. There are of two types: workers and managers.

## **Generality**

Components are first-class values. They can be passed as parameters in the headers of manifolds and atomic processes in order to define composite components.

## **Binding Technology**

Computational processes can be composed by defining in the manager process explicitly the connections between the different incoming and outgoing ports of those components. This form of composition is purely dynamic. Nevertheless, another form of composition is also used when defining composite components. Composite components can be defined by passing components (processes) by parameter at the specification time. They can be used at black-box components that provide and require certain informations to process.

## **Semantics**

No formal semantics.

[ [Top](#) | [Summary](#) | [Component Model](#) | [Usage](#) | [Discussion](#) | [References](#) ]

---

## **Usage**

[ [Domain](#) | [Architecture](#) | [Specification](#) | [Interoperability](#) | [Process](#) ]

## **Application Domain**

Supports asynchronous communication exclusively. Its model is more oriented to control-oriented systems software. Coordination is reduced to the management of communication and synchronization aspects between computational components.

## **Architecture**

It is possible to define new manager process. Nevertheless the

architecture is fix. There are two kind of processes, and there is a set of aspects that can be defined, basically it concerns the creation of process and the management of communications between computational processes. All this following a predefined model.

## **Specification Medium**

Components are specified using the Manifold language.

## **Interoperability and Distribution**

No support.

## **Software Process**

Visifold: A Visual Environment for a Coordination Language [3].

[ [Top](#) | [Summary](#) | [Component Model](#) | [Usage](#) | [Discussion](#) | [References](#) ]

---

## **Discussion**

The significant characteristics of the IWIM model include composibility, which it inherits from data-flow, anonymous communication, and separation of computation concerns from communication concerns. Its disadvantage is its limitation represent a unique coordination model. Not all coordination problems are well-suited to a particular paradigm of coordination.

[ [Top](#) | [Summary](#) | [Component Model](#) | [Usage](#) | [Discussion](#) | [References](#) ]

---

## **References**

1. F. Arbad, ``Coordination of massively concurrent activities,'' CS-R9565, Centrum voor Wiskunde en Informatica (CWI), 1995. File: <ftp://www.cwi.nl/pub/manifold/CS-R9565.ps.Z>
2. F. Arbad, ``The IWIM Model for Coordination of Concurrent Activities,'' *Proceedings of COORDINATION'96*, P. Ciancarini and Chris Hankin (Ed.), LNCS 1061, Springer-Verlag, Cesena, Italy, 1996, pp. 34-55.
3. Pascal Bouvry and Farhad Arbad, ``Visifold\*: A Visual Environment for a Coordination Language,'' *Proceeding of COORDINATION'96*, Paolo Ciancarini and Chris Hankin (Ed.), LNCS 1061, Springer-Verlag, April 1996, pp. 403-406.

[ [Top](#) | [Summary](#) | [Component Model](#) | [Usage](#) | [Discussion](#) | [References](#) ]

---

[ [Front Page](#) | [Index](#) ]